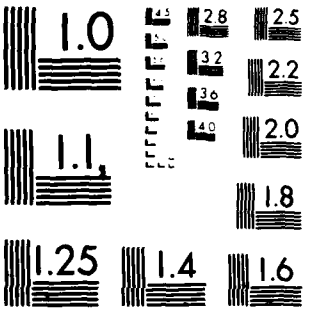
AD-A131 654 A DEDUCTIVE APPROACH TO PROGRAMMING METHODOLOGY(U) 1/1
STANFORD UNIV CA DEPT OF COMPUTER SCIENCE Z MANNA
DEC 81 N00014-76-C-0687

UNCLASSIFIED F/G 9/2 NL

END
DATE FILMED
9 83
DTIC



SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: -- 2. GOVT ACCESSION NO.: AD-A131 654	3. RECIPIENT'S CATALOG NUMBER:
4. TITLE (and Subtitle): Final Report: " A Deductive Approach to Programming Methodology"	5. TYPE OF REPORT & PERIOD COVERED: Final Technical; 1/80 - 12/81 6. PERFORMING ORG. REPORT NUMBER:
7. AUTHOR(s): Professor Zohar Manna	8. CONTRACT OR GRANT NUMBER(s): N00014-76-C-0687
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Department of Computer Science **Stanford** University Stanford, CA 94305	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
11. CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research Information Systems Program 800 North Quincy, Arlington, VA 22217	12. REPORT DATE: 13. NUMBER OF PAGES: 3
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report)

Scientific Officer (1); ONR Branch Office (1); ACO (1); NRL Code 2627 (6); ONR Code 102MR (6); DDC (12).

APPROVED FOR PUBLIC RELEASE
DISTRIBUTION UNLIMITED

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

OFFICE OF NAVAL RESEARCH
GRANT N00014-76-C-0687-FINAL TECHNICAL REPORT
JANUARY 1980-DECEMBER 1981

This report

~~Our~~ research ~~was~~ concentrated on the following topics:

a) Verification of Concurrent programs: The Temporal Framework ([1]).

We first introduce temporal logic as a tool for reasoning about sequences of states. Models of concurrent programs based both on transition graphs and on linear-text representations are presented and the notions of concurrent and fair executions are defined.

The general temporal language is then specialized to reason about those execution sequences that are fair computations of a concurrent program. Subsequently, the language is used to describe properties of concurrent programs.

The set of interesting properties is classified into *invariance* (safety), *eventuality* (liveness), and *precedence* (until) properties. Among the properties studied are: partial correctness, global invariance, clean behavior, mutual exclusion, absence of deadlock, termination, total correctness, intermittent assertions, accessibility, responsiveness, safe liveness, absence of unsolicited response, fair responsiveness, and precedence.

b) Verification of Concurrent Programs: Temporal Proof Principles ([2]).

Here, we present temporal proof methods for establishing properties of concurrent programs. We consider three classes of properties: invariances, eventualities (liveness properties) and precedence (*until* properties).

The proof principle for establishing invariance properties is based on computational induction, and is a generalization of the *inductive assertions* method. For a restricted class of programs we present an algorithm for the automatic derivation of invariant assertions.

In order to establish eventuality properties we present several principles which translate the structure of the program into basic temporal statements about its behavior. These principles can be viewed as providing the temporal semantics of the program. The basic statements thus derived are then combined into temporal proofs for the estblishment of eventuality properties. This method generalizes the method of *intermittent assertions*.

An *until* property is shown to be essentially a combination of a conditional invariance and an eventuality. Consequently the proof method for establishing an until property is a generalization of the method for establishing eventualities.

All the methods are applied to examples.

c) Verification of Sequential Programs: Temporal Axiomatization ([3]).

Earlier, we introduced temporal logic as a tool for reasoning about concurrent programs and specifying their properties ([1]) and presented proof principles for establishing these properties ([2]). Here, we restrict ourselves to deterministic, sequential programs. We present a proof system in which properties of such programs, expressed as temporal formulas, can be proved formally.

Our proof system consists of three parts: a *general part* elaborating the properties of temporal logic, a *domain part* giving an axiomatic description of the data domain, and a *program part* giving an axiomatic description of the program under consideration.

We illustrate the use of the proof system by giving two alternative formal proofs of the total correctness of a simple program.

d) Synthesis of Communicating Processes from Temporal Specifications ([4]).

We apply Propositional Temporal Logic (PTL) to the specification and synthesis of the synchronization part of communicating processes. To specify a process, we give a PTL formula that describes its sequence of communications. The synthesis is done by constructing a model of the given specifications using a tableau-like satisfiability algorithm for PTL. This model can then be interpreted as a program.

e) Deductive Synthesis of the Unification Algorithm ([5]).

The *deductive approach* is a formal program construction method in which the derivation of a program from a given specification is regarded as a theorem-proving task. To construct a program whose output satisfies the conditions of the specification, we prove a theorem stating the existence of such an output. The proof is restricted to be sufficiently constructive so that a program computing the desired output can be extracted directly from the proof. The program we obtain is applicative and may consist of several mutually recursive procedures. The proof constitutes a demonstration of the correctness of this program.

To exhibit the full power of the deductive approach, we apply it to a nontrivial example — the synthesis of a *unification algorithm*. Unification is the process of finding a common instance of two expressions. Algorithms to perform unification have been central to many theorem-proving systems and to some programming-language processors.

The task of deriving a unification algorithm automatically is beyond the power of existing program synthesis systems. In this paper we use the deductive approach to derive an algorithm from a simple, high-level specification of the unification task. We will identify some of the capabilities required of a theorem-proving system to perform this derivation automatically.

f) Special Relations in Program Synthetic Deduction ([6]).

Program synthesis is the automated derivation of a computer program from a given specification. In the *deductive approach*, the synthesis of a program is regarded as a theorem-proving problem; the desired program is constructed as a by-product of the proof. This paper presents a formal deduction system for program synthesis, with special features for handling equality, the equivalence connective, and ordering relations.

In proving theorems involving the equivalence connective, it is awkward to remove all the quantifiers before attempting the proof. The system therefore deals with ***partially skolemized sentences***, in which some of the quantifiers may be left in place. A rule is provided for removing individual quantifiers when required after the proof is under way.

The system is also *nonclausal*; i.e., the theorem does not need to be put into conjunctive normal form. The equivalence, implication, and other connectives may be left intact.

Publications

[1] Z. Manna, A. Pnueli, "Verification of Concurrent Programs: The Temporal Framework", in *The Correctness Problem in Computer Science* (R. S. Boyer and J S. Moore, eds.), International Lecture Series in Computer Science, Academic Press, London, 1981.

[2] Z. Manna, A. Pnueli, "The Temporal Verification of Concurrent Programs: Temporal Proof Principles", *Proc. of the Workshop on Logics of Programs (Yorktown-Heights, NY)*, Springer-Verlag Lecture Notes in Computer Science, 1981.

[3] Z. Manna, "Verification of Sequential Programs: Temporal Axiomatization", in *Theoretical Foundations of Programming Methodology* (F. L. Bauer, E. W. Dijkstra, and C. A. R. Hoare, eds.), NATO Scientific Series, D. Reidel Pub. Co., Holland, 1981.

[4] Z. Manna, P. Wolper, "Synthesis of Communicating Processes from Temporal Specifications" *Proceedings of the Workshop on Logics of Programs (Yorktown-Heights)*, NY, Springer-Verlag Lecture Notes in Computer Science, 1981.

[5] Z. Manna, R. Waldinger, "Deductive Synthesis of the Unification Algorithm", *Science of Computer Programming* 1 (1981), pp. 5–48.

[6] Z. Manna, R. Waldinger, "Special Relations in Program Synthetic Deductions", Technical Report, Computer Science Department, Stanford University, April 1982.